AP® COMPUTER SCIENCE A
2015 GENERAL SCORING GUIDELINES

Apply the question assessment rubric first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times, or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

(v) Array/collection access confusion (`[] get`)

(w) Extraneous code that causes side effect (*e.g., writing to output, failure to compile*)

(x) Local variables used but none declared

(y) Destruction of persistent data (*e.g., changing value referenced by parameter*)

(z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side effect (*e.g., precondition check, no-op*)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` and vice versa
- `length/size` confusion for array, `String`, `List`, or `ArrayList`, with or without `( )`
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration (*e.g.,* `int[size] nums = new int[size];`)
- Missing `;` where structure clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context; for example, "`ArayList`" instead of "`ArrayList`". As a counterexample, note that if the code declares "`Bug bug;`", then uses "`Bug.move()`" instead of "`bug.move()`", the context does **not** allow for the reader to assume the object instead of the class.*

AP® COMPUTER SCIENCE A
2015 SCORING GUIDELINES

Question 4: Number Group

Part (a)	**Interface: `NumberGroup`**	**2 points**

Intent: *Define interface to represent a number group*

+1 `interface NumberGroup` (*point lost if visibility private*)

+1 `boolean contains(int num);`
(*point lost if visibility not public or extraneous code present*)

Part (b)	**Class: `Range`**	**5 points**

Intent: *Define implementation of* `NumberGroup` *representing a range of numbers*

+1 `class Range implements NumberGroup` (*point lost if visibility private*)

+1 Declares appropriate `private` instance variable(s)

+1 Uses correct constructor header

+1 Initializes instance variables within constructor using parameters
(*point lost if bounds errors occur in container use*)

+1 Computes and returns correct value from `contains`
(*point lost for incorrect method header*)

Part (c)	`contains`	**2 points**

Intent: *Determine whether integer is part of any of the member number groups*

+1 Calls `contains` on elements of `groupList` in context of loop *(no bounds errors)*

+1 Computes and returns correct value

Question-Specific Penalties

-1 (s) Inappropriate use of `static`

AP® COMPUTER SCIENCE A
2015 CANONICAL SOLUTIONS

Question 4: Number Group

Part (a):

```
public interface NumberGroup
{
    boolean contains(int num);
}
```

Part (b):

```
public class Range implements NumberGroup
{
    private int min;
    private int max;

    public Range(int min, int max)
    {
        this.min=min;
        this.max=max;
    }

    public boolean contains(int num){
        return num >= min && num <= max;
    }
}
```

Part (c):

```
public boolean contains(int num){
       for (NumberGroup group : groupList){
           if (group.contains(num)){
               return true;
           }
       }
       return false;
}
```

4Aa

1of3

4. This question involves the design of an interface, writing a class that implements the interface, and writing a method that uses the interface.

(a) A *number group* represents a group of integers defined in some way. It could be empty, or it could contain one or more integers.

Write an interface named `NumberGroup` that represents a group of integers. The interface should have a single `contains` method that determines if a given integer is in the group. For example, if `group1` is of type `NumberGroup`, and it contains only the two numbers `-5` and `3`, then `group1.contains(-5)` would return `true`, and `group1.contains(2)` would return `false`.

Write the complete `NumberGroup` interface. It must have exactly one method.

```
Public interface Number Group
{

    boolean Contains(int num);

}
```

Part (b) begins on page 21.

(b) A *range* represents a number group that contains all (and only) the integers between a minimum value and a maximum value, inclusive.

Write the `Range` class, which is a `NumberGroup`. The `Range` class represents the group of `int` values that range from a given minimum value up through a given maximum value, inclusive. For example, the declaration

```
NumberGroup range1 = new Range(-3, 2);
```

represents the group of integer values -3, -2, -1, 0, 1, 2.

Write the complete `Range` class. Include all necessary instance variables and methods as well as a constructor that takes two `int` parameters. The first parameter represents the minimum value, and the second parameter represents the maximum value of the range. You may assume that the minimum is less than or equal to the maximum.

```
public Class Range implements Number Group
{
    int[] range;
    public Range(int min, int max)
    { int i=0;
      int total = Math.abs(min - max-1);
       range = new int[total];
       for(int X= min; X < max+1; X++){
         range[i] = X;
         i++;
       }
    }
    Public boolean Contains(int num)
    { boolean in=false;
        for( int a: range){
          if( a== num)
              in = true;}
        return in;} }
```

Part (c) begins on page 22.

GO ON TO THE NEXT PAGE.

Complete method `contains` below.

```
/** Returns true if at least one of the number groups in this multiple group contains num;
 *          false otherwise.
 */
public boolean contains(int num)
```

```
{
  boolean in = false;
  for( Range a : groupList) {
      if ( a.contains(num))
          in = true;
      }
  return in;
}
```

4Ba

1of3

4. This question involves the design of an interface, writing a class that implements the interface, and writing a method that uses the interface.

(a) A *number group* represents a group of integers defined in some way. It could be empty, or it could contain one or more integers.

Write an interface named `NumberGroup` that represents a group of integers. The interface should have a single `contains` method that determines if a given integer is in the group. For example, if `group1` is of type `NumberGroup`, and it contains only the two numbers `-5` and `3`, then `group1.contains(-5)` would return `true`, and `group1.contains(2)` would return `false`.

Write the complete `NumberGroup` interface. It must have exactly one method.

```
Public Interface NumberGroup
{
    Public boolean contains()
    {
      return true;
    }
}
```

Part (b) begins on page 21.

(b) A *range* represents a number group that contains all (and only) the integers between a minimum value and a maximum value, inclusive.

Write the `Range` class, which is a `NumberGroup`. The `Range` class represents the group of `int` values that range from a given minimum value up through a given maximum value, inclusive. For example, the declaration

```
NumberGroup range1 = new Range(-3, 2);
```

represents the group of integer values -3, -2, -1, 0, 1, 2.

Write the complete `Range` class. Include all necessary instance variables and methods as well as a constructor that takes two `int` parameters. The first parameter represents the minimum value, and the second parameter represents the maximum value of the range. You may assume that the minimum is less than or equal to the maximum.

```
public class Range implements NumberGroup
{
  private int[] group;
  private int first;
  private int last;
  public Range(int min, int max)
  {
    first = min;
    last = max;
    for(int i=first; i<=last; i++)
      group[i] = first + i;
  }
```

Part (c) begins on page 22.

4Bc

3 of 3

Complete method `contains` below.

```
/** Returns true if at least one of the number groups in this multiple group contains num;
 *          false otherwise.
 */
public boolean contains(int num){
for(int i=0; i<grouplist.size(); i++)
{
    if(grouplist.get(i).contains(num))
        return true;
}
return false
}
```

4Ca
1of3

4. This question involves the design of an interface, writing a class that implements the interface, and writing a method that uses the interface.

(a) A *number group* represents a group of integers defined in some way. It could be empty, or it could contain one or more integers.

Write an interface named `NumberGroup` that represents a group of integers. The interface should have a single `contains` method that determines if a given integer is in the group. For example, if `group1` is of type `NumberGroup`, and it contains only the two numbers `-5` and `3`, then `group1.contains(-5)` would return `true`, and `group1.contains(2)` would return `false`.

Write the complete `NumberGroup` interface. It must have exactly one method.

```
public interface Number group {
    public boleam contains() {
    IF (group1 == Number Group) {
```

Part (b) begins on page 21.

4Cb

(b) A *range* represents a number group that contains all (and only) the integers between a minimum value and a maximum value, inclusive.

2of3

Write the `Range` class, which is a `NumberGroup`. The `Range` class represents the group of `int` values that range from a given minimum value up through a given maximum value, inclusive. For example, the declaration

```
NumberGroup range1 = new Range(-3, 2);
```

represents the group of integer values -3, -2, -1, 0, 1, 2.

Write the complete `Range` class. Include all necessary instance variables and methods as well as a constructor that takes two `int` parameters. The first parameter represents the minimum value, and the second parameter represents the maximum value of the range. You may assume that the minimum is less than or equal to the maximum.

```
public class Range {
    private int minValue;
    private int maxValue;

public Range(minValue, maxValue) {
    min = minValue;
    max = maxValue;
}
```

Part (c) begins on page 22.

4Cc

3of3

Complete method `contains` below.

```
/** Returns true if at least one of the number groups in this multiple group contains num;
 *          false otherwise.
 */
public boolean contains(int num) {

}
```

```
return true;

else {
    return false;
}
```

AP® COMPUTER SCIENCE A
2015 SCORING COMMENTARY

Question 4

Overview

This question asked students to write an interface specification, a complete class implementing that interface, and the body of a method in another class implementing the interface. In part (a) students were asked to write the `NumberGroup` interface. A `NumberGroup` represents a collection of integers. Its single required method is `contains`, which takes as its parameter an integer and returns `true` when that integer is part of the collection and `false` otherwise. The students needed to know the correct syntax for the interface declaration and how to provide exactly one method declaration inside that interface. It is particularly important that the students not attempt to provide any method implementation.

In part (b) students were asked to write a `Range` class that implements the `NumberGroup` interface. Students were expected to provide a constructor with two integer parameters and private instance variables to maintain enough object state for the `contains` method to work. Two kinds of approaches work equally well here: saving the minimum and maximum values of the range; or building an array or `ArrayList` containing all the elements of the range. Students also needed to write the `contains` method that was specified by the `NumberGroup` interface.

In part (c) students were asked to implement the `contains` method of the `MultipleGroups` class. The `MultipleGroups` class has a single `List` instance variable `groupList`, which is used to store a collection of `NumberGroup` objects. The method takes an integer and returns `true` if and only if the integer is contained in one or more of the number groups in `groupList`.

Sample: 4A
Score: 8

In part (a) the student writes a correct solution. The interface header is correct, earning the first point. The `contains` method is declared to take a single `int` parameter and to return `boolean`, earning the second point. The student earned 2 points in part (a).

In part (b) the student writes the correct class header, earning the first point. The array instance variable `range` is appropriate but is not declared `private`, so the second point is not earned. The constructor header is correct, with two `int` parameters, earning the third point. The student's formula for determining the correct array length (`total`) is correct, because whenever `max ≥ min`, as assumed in the problem statement, `Math.abs(min - max - 1)` is equal to `max - min + 1`. This is the correct number of entries. The `range` array is correctly initialized and populated with all the numbers in the range, earning the fourth point. In the `contains` method, a local variable `in` is initialized to `false`. Next, an iteration over the elements of the `range` array sets `in` to `true` whenever `num` is found. After the loop terminates, the value `in` is returned. Therefore, the fifth point is earned. The student earned 4 points in part (b).

In part (c) the student initializes a `boolean` variable `in` to `false`. Next, an iteration over elements of `groupList` proceeds. For each element, the `contains` method is correctly called, earning the first point. The variable `in` is set to `true` when `contains` returns `true`. After the loop, the value of `in` is returned. This is the correct logic, so the second point is earned. The student earned 2 points in part (c).

Question 4 (continued)

Sample: 4B
Score: 6

In part (a) the student provides the correct `NumberGroup` interface header, earning the first point. The `contains` method header is incorrect, missing the required `int` parameter. Therefore, the student does not earn the second point. Note that even if the method header had been correct, the student attempts to implement the method inside the interface. This is not allowed, so the student could not have earned the second point. The student earned 1 point in part (a).

In part (b) the student provides the correct `Range` class header, earning the first point. The `private` instance variables are also correctly declared, earning the second point. The constructor header is correct, having two `int` parameters. Therefore, the student earned the third point. In order to earn the fourth point, the student needs to correctly initialize the instance variables from the parameters. The first two assignments are correct, but the loop to initialize `group` fails because `group` was never instantiated. Thus, the student does not earn the fourth point. Because there is no implementation of the `contains` method, the fifth point is not earned. The student earned 3 points in part (b).

In part (c) the student correctly iterates over each valid index of `groupList`. At each position, the `NumberGroup` is accessed, and its `contains` method is called. This earned the first point. In the loop, if any `NumberGroup` in `groupList` is found to contain `num`, `true` is immediately returned. Should the loop terminate normally, it is necessarily the case that no such `NumberGroup` was found, and `false` is returned. This correct logic earned the second point. The student earned 2 points in part (c).

Sample: 4C
Score: 2

In part (a) the student writes the correct `NumberGroup` interface header, earning the first point. The `contains` method header is incorrect because it is missing the required `int` parameter. Also, the student begins an implementation, which is not allowed in an interface definition. For either of these reasons, the second point is not earned. The student earned 1 point in part (a).

In part (b) the student omits "`implements NumberGroup`" from the `Range` class header, so the first point is not earned. The student declares appropriate `private` instance variables and the second point is earned. The constructor header is incorrect because the `int` types for the two parameters are missing. Thus, the third point is not earned. The fourth point for initialization is also not earned because the instance variables are not assigned correctly. The student does not provide a `contains` method and does not earn the fifth point. The student earned 1 point in part (b).

In part (c) the student neither writes a loop nor calls the `contains` method. For these reasons, the student earned no points in part (c).